Geosci. Model Dev., 11, 4621–4635, 2018
https://doi.org/10.5194/gmd-11-4621-2018

Geoscientific Model Development
Open Access
EGU

The VOLNA-OP2 tsunami code (version 1.5)

Istvan Z. Reguly[1], Daniel Giles[2], Devaraj Gopinathan[3], Laure Quivy[4], Joakim H. Beck[5], Michael B. Giles[6], Serge Guillas[3], and Frederic Dias[2]

[1]Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, Prater u 50/a, 1088 Budapest, Hungary
[2]School of Mathematics and Statistics, University College Dublin, Dublin, Ireland
[3]Department of Statistical Science, University College London, London, UK
[4]Centre de Mathématiques et de Leurs Applications (CMLA), Ecole Normale Supérieure, Paris-Saclay, Centre National de la Recherche Scientifique, Université Paris-Saclay, 94235 Cachan, France
[5]Computer, Electrical and Mathematical Science and Engineering Division (CEMSE), King Abdullah University of Science and Technology (KAUST), Thuwal, 23955-6900, Saudi Arabia
[6]Math Institute, University of Oxford, Oxford, UK

Correspondence: Istvan Z. Reguly (reguly.istvan@itk.ppke.hu)

Received: 24 January 2018 – Discussion started: 8 March 2018
Revised: 21 October 2018 – Accepted: 24 October 2018 – Published: 19 November 2018

Abstract. In this paper, we present the VOLNA-OP2 tsunami model and implementation; a finite-volume nonlinear shallow-water equation (NSWE) solver built on the OP2 domain-specific language (DSL) for unstructured mesh computations. VOLNA-OP2 is unique among tsunami solvers in its support for several high-performance computing platforms: central processing units (CPUs), the Intel Xeon Phi, and graphics processing units (GPUs). This is achieved in a way that the scientific code is kept separate from various parallel implementations, enabling easy maintainability. It has already been used in production for several years; here we discuss how it can be integrated into various workflows, such as a statistical emulator. The scalability of the code is demonstrated on three supercomputers, built with classical Xeon CPUs, the Intel Xeon Phi, and NVIDIA P100 GPUs. VOLNA-OP2 shows an ability to deliver productivity as well as performance and portability to its users across a number of platforms.

1 Introduction

After the Indian Ocean tsunami of 26 December 2004, Bernard et al. (2006) emphasised that one of the greatest contributions of science to society is to serve it purposefully, as when providing forecasts to allow communities to respond before a disaster strikes. In the last 12 years, the numerical modelling of tsunamis has experienced great progress – see Behrens and Dias (2015). There is a variety of mathematical models, such as shallow-water equations (see Titov and Gonzalez (1997), Liu et al. (1998), Gailler et al. (2013), Zhang and Baptista (2008), Macías et al. (2017), and Dutykh et al. (2011)), the Boussinesq equations (see Kennedy et al. (2000) and Lynett et al. (2002)), or the Navier–Stokes equations (see (Abadie et al., 2012) and Gisler et al. (2006)) and a large number of implementations, primarily for individual target computer architectures. The use cases of such models are wide ranging, and most rely on high numerical accuracy as well as high computational performance to deliver results – examples include sensitivity analysis by Goda et al. (2014), probabilistic tsunami hazard assessments by Geist and Parsons (2006), Davies et al. (2017), and Anita et al. (2017), and more efficient and informed tsunami early warning by Yusuke et al. (2014) and Castro et al. (2015).

For widespread use three key ingredients are needed; first, the stability and robustness of the numerical approach, which gives a confidence in the results produced; second, the computational performance of the code, which allows for obtaining the right results quickly, efficiently utilising the available computational resources; and third, the ability to integrate into a workflow, allowing for simple preprocessing and postprocessing, efficiently supporting the kinds of use cases that come up – for example large numbers of different initial conditions.

In Sect. 2 we discuss a number of codes currently being used in production, which as such are trusted and reliable codes, as part of a workflow. Yet, the computational performance of most of these codes is "good enough"; they were written by domain scientists and may have been tuned to one architecture or another, but, for example, GPU support is almost non-existent. In today's and tomorrow's quickly changing hardware landscape, however, "future-proofing" numerical codes is of exceptional importance for continued scientific delivery. Domain scientists can not be expected to keep up with architectural advances and spend a significant amount of time re-factoring code to new hardware. *What* to compute must be separated from *how* it is computed – indeed in a recent paper by Lawrence et al. (2018), leaders in the weather community chart the ways forward and point to domain-specific languages (DSLs) as a potential way to address this issue.

OP2, by Mudalige et al. (2012), is such a DSL, embedded in C/C++ and Fortran; it has been in development since 2009. It provides an abstraction for expressing unstructured mesh computations at a high level and then provides automated tools to translate scientific code written once into a range of high-performance implementations targeting multicore central processing units (CPUs), graphics processing units (GPUs), and large heterogeneous supercomputers. The original VOLNA model (Dutykh et al., 2011) was already discussed and validated in detail – it was used in production for small-scale experiments and modelling but was inadequate for targeting large-scale scenarios and statistical analysis; therefore it was re-implemented on top of OP2; this paper describes the process, challenges, and results from that work.

As VOLNA-OP2 delivered a qualitative leap in terms of possible uses due to the high performance it can deliver on a variety of hardware architectures, its users have started integrating it into a wide variety of workflows; one of the key uses is for uncertainty quantification: for the stochastic inversion problem of the 2004 Sumatra tsunami in Gopinathan et al. (2017), for developing Gaussian process emulators that help reduce the number of simulation runs in Beck and Guillas (2016) and Liu and Guillas (2017), applications of stochastic emulators to a submarine slide at the Rockall Bank in Salmanidou et al. (2017), a study of run-up behind islands in Stefanakis et al. (2014), the durability of oscillating wave surge converters when hit by tsunamis in O'Brien et al. (2015), tsunamis in the St. Lawrence Estuary in Poncet et al. (2010), a study of the generation and inundation phases of tsunamis in Dias et al. (2014), and others.

The time dependency in the deformation enables the tsunami to be actively generated – see Dutykh and Dias (2009). This is a step forward from the common passive mode of tsunami genesis that utilises an instantaneous rupture. The active mode is particularly important for tsunamigenic earthquakes with long and slow ruptures, e.g. the 2004 Sumatra–Andaman event described in Lay et al. (2005) and Gopinathan et al. (2017), and submerged landslides in Løvholt et al. (2015), e.g. the Rockall Bank event in Salmanidou et al. (2017).

These applications present a number of challenges in integration into the workflow, as well as scalable performance: the need for extracting snapshots of state variables on the full mesh, or at a number of specified locations, and capturing the maximum wave elevation or inundation – all in the context of distributed memory execution.

As the above references indicate, VOLNA-OP2 has already been key in delivering scientific results in a range of scenarios, and through the collaboration of the authors, it is now capable of efficiently supporting a number of use cases, making it a versatile tool to the community; therefore we have now publicly released it: it is freely available at https://github.com/reguly/volna (last access: 11 November 2018).

The rest of the paper is organised as follows: Sect. 2 discusses related work; Sect. 3 presents the OP2 library, upon which VOLNA-OP2 is built; Sect. 4 discusses the VOLNA simulator itself, its structure, and features; Sect. 5 discusses performance and scalability results on CPUs and GPUs; and finally Sect. 6 draws conclusions.

2 Related work

Tsunamis have long been a key target for scientific simulations. Behrens and Dias (2015) give a detailed look at various mathematical, numerical, and implementation approaches to past and current tsunami simulations. The most common set of equations solved are the shallow-water equations, and most codes use structured and nested meshes. A popular discretisation is finite differences, such codes include NOAA's MOST (Titov and Gonzalez, 1997), COMCOT (Liu et al., 1998), and CENALT (Gailler et al., 2013). On more flexible meshes many codes, such as SELFE (Zhang and Baptista, 2008), TsunAWI (Harig et al., 2008), ASCETE (Vater and Behrens, 2014), and Firedrake-Fluids (Jacobs and Piggott, 2015), use the finite-element discretisation or the finite-volume discretisation in the cases of the VOLNA code (Dutykh et al., 2011), GeoClaw (George and LeVeque, 2006), or HySEA (Macías et al., 2017). Another model is described by the Boussinesq equations – these equations and the solver are more complex than shallow-water solvers. Since they are primarily needed only for dispersion (see Glimsdal et al. (2013)), they are used less commonly; examples include FUNWAVE (Kennedy et al., 2000) and COULWAVE (Lynett et al., 2002). Finally, the 3-D Navier–Stokes equations provide the most complete description, but they are significantly more complex than other models – examples include SAGE (Gisler et al., 2006) and the work of Abadie et al. (2012).

Most of these codes described above work on CPUs, and while there has been some work on GPU implementations by Satria et al. (2012), Liang et al. (2009a, b), Brodtkorb et al. (2010), and Acuña and Aoki (2009), who use structured

meshes and finite differences or finite volumes, it is unclear whether these are used in production, and they are not open source. Celeris (Tavakkol and Lynett, 2017) is a Boussinesq solver that uses finite volumes and a structured mesh – it is hand-coded for GPUs using graphics shaders, and its source code is available; however it can only use a single GPU.

As far as we are aware, only Tsunami-HySEA (Macías et al., 2017), which also uses finite volumes, uses GPU clusters in production – that code, however, only supports GPUs and is hand-written in CUDA. Performance reported by Castro et al. (2015) on a 10 million point test case shows a strong scaling efficiency going from 1 to 12 GPUs between 88 % and 73 % (overall 12 GPUs are 5.88 faster than 1 GPU) and a 25× speed-up with 1 GPU over an unspecified (likely single core) CPU implementation. Direct comparison to VOLNA-OP2 is not possible since Tsunami-HySEA uses (nested) structured meshes, and the multi-GPU version is not open source.

3 The OP2 domain-specific language

The OP2 library (Mudalige et al., 2012) is a DSL embedded in C and Fortran that allows unstructured mesh algorithms to be expressed at a high level and provides automatic parallelisation and a number of other features. It provides an abstraction that lets the domain scientist describe a mesh using a number of sets (such as quadrilaterals or vertices), connections among these sets (such as edges to nodes), and data defined on sets (such as x and y coordinates on vertices). Once the mesh is defined, an algorithm can be implemented as a sequence of parallel loops, each over all elements of a given set applying different "kernel functions", accessing data either directly on the iteration set or indirectly through, at most, one level of indirection. This abstraction enables the implementation of a wide range of algorithms, such as the finite-volume algorithms that VOLNA uses, but it does require that for any given parallel loop, the order of execution must not affect the end result (within machine precision) – this precludes the implementation of Gauss–Seidel iterations, for example.

OP2 enables its users to write an application only once using its API, which is then automatically parallelised to utilise multicore CPUs, GPUs, and large supercomputers through the use of MPI, OpenMP, and CUDA. This is carried out in part through a code generator that parses the parallel loop expressions and generates code around the computational kernel to facilitate parallelism and data movement, and in part through different back-end libraries that manage data, including MPI halo exchanges, or GPU memory management, as shown in Fig. 1. For more details, see Giles et al. (2011) and Mudalige et al. (2012).

3.1 Parallelisation approaches in OP2

OP2 takes full responsibility for orchestrating parallelism and data movement – from the user perspective, the code written looks and feels like sequential C code that makes calls to an external library. To utilise clusters and supercomputers, OP2 uses the message passing interface (MPI) to parallelise in a distributed memory environment; once the mesh is defined by the user, OP2 automatically partitions and distributes it among the available resources. It uses the standard owner-compute approach with halo exchanges and overlaps computations with communications. In conjunction with MPI, OP2 uses a number of shared-memory parallelisation approaches, such as CUDA and OpenMP.

A key challenge in the finely grained parallelisation of unstructured mesh algorithms is the avoidance of race conditions when data are indirectly modified. For example, in a parallel loop over edges, when indirectly incrementing data on vertices, multiple edges may try to increment the same vertex, leading to race conditions. OP2 uses a colouring approach to resolve this; elements of the iteration set are grouped into mini-partitions, and each element within these mini-partitions is coloured, so no two elements of the same colour access the same value indirectly. Subsequently, mini-partitions are coloured as well. For CUDA, we assign mini-partitions of the same colour to different CUDA thread blocks, and for OpenMP to different threads. There is then a global synchronisation among different mini-partition colours. In the case of CUDA, threads processing elements within each thread block use the first level of colouring to apply increments in a safe way, with block-level synchronisation in between. Code generation that is suitable for auto-vectorisation by the compilers is also supported; it carries out the packing and unpacking of vector registers. The results obtained on different architectures may only differ due to differences in compiler optimisations (particularly at aggressive levels) and the different order in which partial results are accumulated. Previous work describes further details, accuracy, and performance comparisons of various architectures; these are available in Mudalige et al. (2012) and Reguly et al. (2007).

3.2 Input and output

OP2 supports parallel file I/O through the HDF5 library (The HDF Group, 2000–2010), which is critically important to its integration into VOLNA's workflow: reading in the input problem and writing out data required for analysis simultaneously on multiple processes.

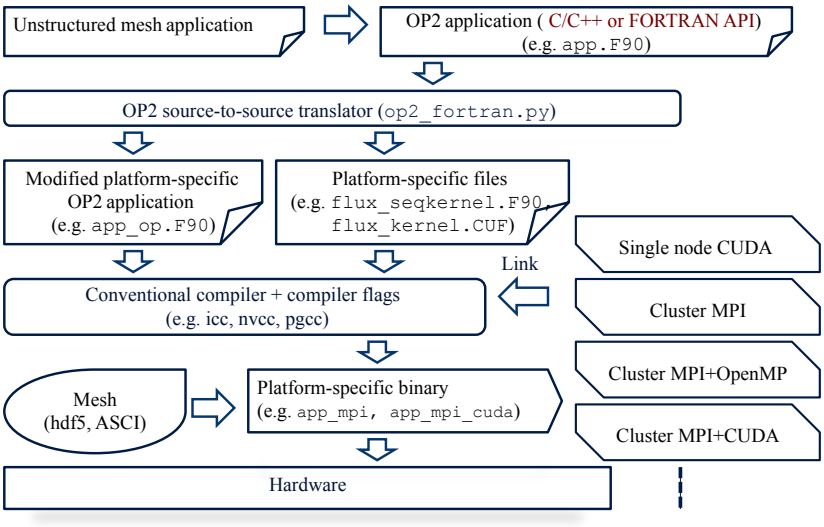

Figure 1. Building system with OP2.

4 The VOLNA simulator

4.1 Model, numerics, and previous validation

The finite-volume framework is the most natural numerical method to solve the non-linear shallow-water equations (NSWEs), in part because of their ability to treat shocks and breaking waves. It belongs to a class of discretisation schemes that are highly efficient in the numerical solution of systems of conservation laws, which are common in compressible and incompressible fluid dynamics. Finite-volume methods are preferred over finite differences and often over finite elements because they intrinsically address conservation issues, improving their robustness: total energy, momentum, and mass quantities are conserved exactly, assuming no source terms and appropriate boundary conditions. The code was validated against the classical benchmarks in the tsunami community as described below.

4.2 Numerical model

Following the needs of the target applications, the following non-dispersive NSWEs (in Cartesian coordinates) form the physical model of VOLNA:

$$H_t + \nabla \cdot (H\boldsymbol{v}) = 0, \tag{1}$$

$$(H\boldsymbol{v})_t + \nabla \cdot \left(H\boldsymbol{v} \otimes \boldsymbol{v} + \frac{g}{2} H^2 \mathbf{I}_2 \right) = gH\nabla d. \tag{2}$$

Here, $d(\boldsymbol{x},t)$ is the time-dependent bathymetry, $\boldsymbol{v}(\boldsymbol{x},t)$ is the horizontal component of the depth-averaged velocity, g is the acceleration due to gravity, and $H(\boldsymbol{x},t)$ is the total water depth. Further, $\mathbf{I}_2$ is the identity matrix of order 2. The tsunami wave height or elevation of free surface $\eta(\boldsymbol{x},t)$ is computed as

$$\eta(\boldsymbol{x},t) = H(\boldsymbol{x},t) - d(\boldsymbol{x},t), \tag{3}$$

where the sum of static bathymetry $d_s(\boldsymbol{x})$ and the dynamic seabed uplift $u_z(\boldsymbol{x},t)$ constitute the dynamic bathymetry,

$$d(\boldsymbol{x},t) = d_s(\boldsymbol{x}) + u_z(\boldsymbol{x},t). \tag{4}$$

d_s is usually sourced from bathymetry datasets pertaining to the region of interest (for example global datasets like ETOPO1/GEBCO or regional bathymetries). The vertical component $u_z(\boldsymbol{x},t)$ of the seabed deformation is calculated depending on the physics of tsunami generation, e.g. via co-seismic displacement for finite fault segmentations by Gopinathan et al. (2017), submarine sliding by Salmanidou et al. (2017, 2018), etc.

In addition to the capabilities of employing active generation and consequent tsunami propagation, VOLNA also models the run-up–run-down (i.e. the final inundation stage of the tsunami). These three functionalities qualify VOLNA to simulate the entire tsunami life cycle. The ability of NSWEs (1)–(2) to model both propagation and run-up and run-down processes was validated in Kervella et al. (2007) and Dutykh et al. (2011), respectively. Thus, the use of a uniform model for the entire life cycle obviates many technical issues such as the coupling between the seabed deformation and the sea surface deformation and the use of nested grids.

VOLNA uses the cell-centred approach for control volume tessellation, meaning that degrees of freedom are associated with cell barycentres. However, in order to improve the spatial accuracy, a second-order extension is employed. A local gradient of the physical variables over each cell is calculated; then a limited linear projection of the variables at the cell interfaces is used within the numerical flux solver. The limiter used is a restrictive version of the scheme purposed by Barth and Jespersen (1989); the minimum calculated limiter of the physical variables within a cell is used in the reconstruction. This limiter ensures that numerical oscillations are con-

strained in realistic cases. A Harten–Lax–van Leer (HLLC) numerical flux which incorporates the contact discontinuity is used to ensure that the standard conservation and consistency properties are satisfied: the fluxes from adjacent triangles that share an edge exactly cancel when summed and the numerical flux with identical state arguments reduces to the true flux of the same state. Details of the numerical implementation can be found in Dutykh et al. (2011).

4.3 Validation

The original version of VOLNA was thoroughly validated against the National Tsunami Hazard Mitigation Program (NTHMP) benchmark problems (Dutykh et al., 2011). A brief look at how the new implementation, which utilises the more restrictive limiter, performs with regards to two benchmark problems is given below. The reader is referred to the original paper (Dutykh et al., 2011) or the NTHMP website for further details on the set-up of the benchmark problems.

4.3.1 Benchmark problem 1 – solitary wave on a simple beach

The analytical solution to the run-up of a solitary wave on a sloping beach was derived by Synolakis (1987). Thus, in this benchmark problem one compares the simulated results with the derived analytical solution.

Set-up

The beach bathymetry comprises a constant depth (d) followed by a sloping plane beach of angle $\beta = \operatorname{arccot}(19.85)$. The initial water level is defined as a solitary wave of height η centred at a distance L from the toe of the beach and the initial wave-particle velocity is proportional to the initial water level:

$$H(x,0) = \eta \operatorname{sech}^2\left(\gamma\left(x - X_1\right)/d\right), \tag{5}$$

$$u(x,0) = -\sqrt{\frac{g}{d}}H. \tag{6}$$

Here $x = X_0 = d\cot(\beta)$, $L = \operatorname{arccosh}(\sqrt{20})/\gamma$, $X_1 = X_0 + L$, and $\gamma = \sqrt{3\eta/4d}$. For this benchmark problem the following ratio must also hold: $\eta t/d = 0.019$.

Tasks

In order to verify the model, the wave run-up at various time steps (Fig. 2) and the wave height at two locations ($x/d = 0.25$ and $x/d = 9.95$) (Fig. 3) are compared to the analytical solution. The test was run on a node of CSD3 Wilkes2 utilising a P100 GPU.

It can be seen from the plots above that the agreement between numerical results and the analytical solutions is very good. Therefore, the new implementation of the model is able to accurately simulate the run-up of the solitary wave.

4.3.2 Benchmark problem 2 – wave run-up onto a complex 3-D beach

This benchmark problem involves the comparison of laboratory results for a tsunami run-up onto a complex 3-D beach with simulated results. The laboratory experiment reproduces the 1993 Hokkaido–Nansei–Oki tsunami, which struck the island of Okushiri, Japan. The experiment is a 1 : 400 scale model of the bathymetry and topography around a narrow gully and the tsunami is an incident wave fed in as a boundary condition.

Set-up

The computational and laboratory domain corresponds to a 5.49 m by 3.40 m wave tank and the bathymetry for the domain is given for 0.014 m by 0.014 m grid cells. The incoming wave is incident on the $x = 0$ m boundary and is defined for the first 22.5 s (Fig. 4a), after which it is recommended that a non-reflective boundary condition be set. At $y = 0$, $y = 3.4$, and $x = 5.5$ m fully reflective boundaries are to be defined.

Tasks

The validation in the model involves comparing the temporal variation of the moving shoreline, the water height at fixed gauges, and the maximum run-up. For the basis of this brief validation, we compared the water height at three gauges installed in the tank, located at (4.521, 1.196), (4.521, 1.696), and (4.521, 2.196).

It can be seen from the gauge plots in Fig. 4b–d that the first elevation wave arrives between 15 and 25 s. The overall dynamics of this elevation wave is accurately captured by the model at all the gauges, particularly the arrival time and initial amplitude. Considering the results of the two benchmark tests and the full validation of the original VOLNA code, one can see that the new implementation, which implements a more restrictive limiter, still preforms satisfactorily and is consistent with the previous version. The benchmark was run on a 24-core Intel(R) Xeon(R) E5-2620 v2 CPU.

4.4 Code structure

The structure of the code is outlined in Algorithm 1; the user inputs a configuration file (.vln), which specifies the mesh to be read in from Gmsh files, as well as initial and boundary conditions of state variables, such as the bathymetry deformation starting the tsunami, which can be defined in various ways (mathematical expressions or files, or a mix of both). We use a variable time step third-order (four stage) Runge–Kutta method for evolving the solution in time. In each iteration, events may be triggered, e.g. further bathymetry deformations, displaying the current simulation time, or outputting simulation data to VTK files for visualisation.

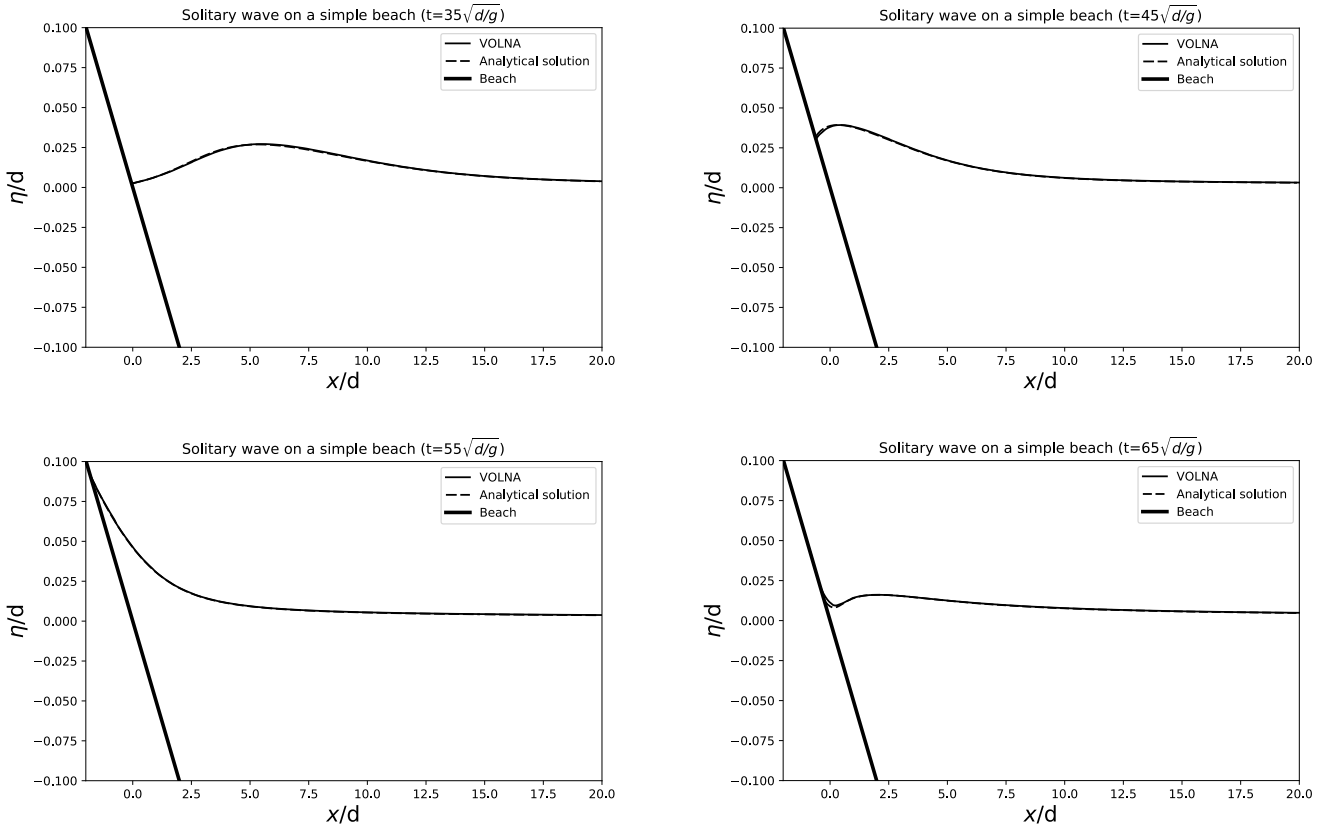

Figure 2. Solitary wave on a simple beach – comparison between the simulated run-up and analytical solution at the shoreline (time = 35, 45, 55, 65 $\sqrt{d/g}$). Solid line – VOLNA; dashed line – analytical solution; thick line – beach.

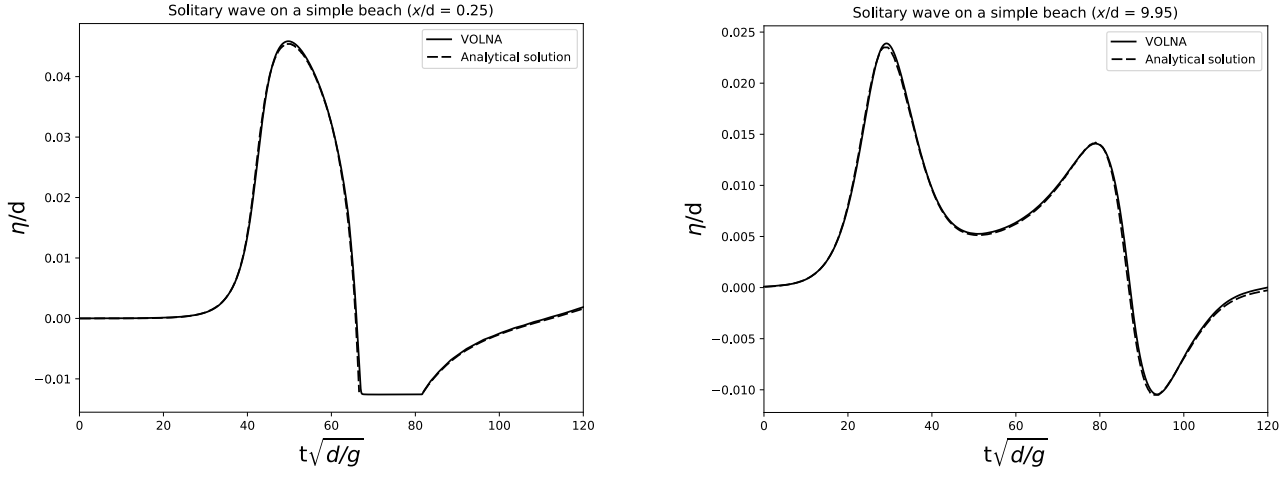

Figure 3. Solitary wave on a simple beach – comparison between VOLNA and solution at different locations: **(a)** $x/d = 0.25$: notice that the location becomes "dry" for $t \approx (67\sqrt{d/g} - 82\sqrt{d/g})$; **(b)** $x/d = 9.95$.

The original VOLNA source code was implemented in C++, utilising libraries such as Boost (Schling, 2011). This gives a very clear structure, abstracting data management, event handling, and low-level array operations for the higher-level algorithm – an example is shown in Fig. 5. While this coding style was good for readability, it had its limitations in terms of performance: there was an excessive amount of data movement and certain operations could not be parallelised – indirect increments with potential race conditions in particular. Some features – such as describing the bathymetry lift

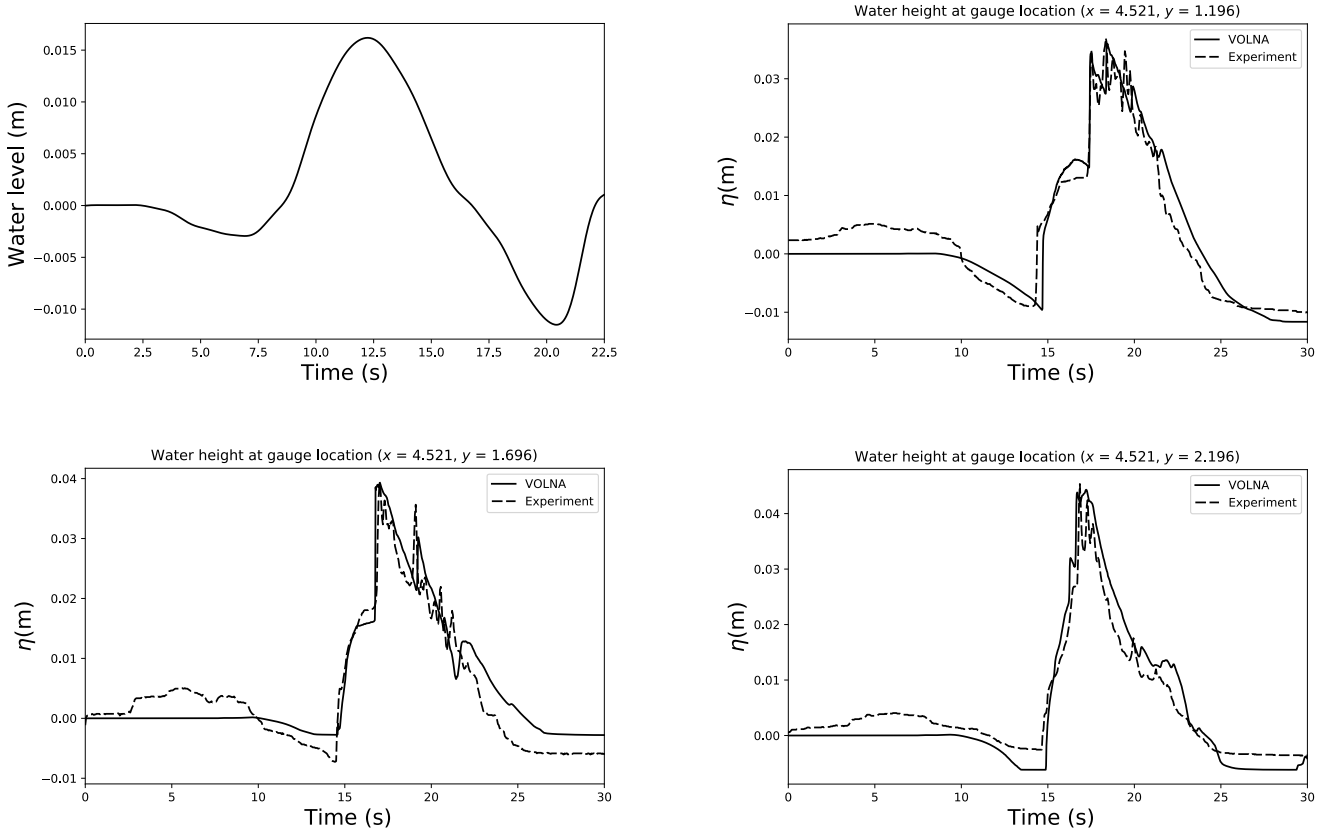

Figure 4. (a) The incoming water level incident on the $x = 0$ m boundary and comparison between VOLNA and laboratory results at different locations for benchmark problem 2: **(b)** $x = 4.521$, $y = 1.196$; **(c)** $x = 4.521$, $y = 1.696$; **(d)** $x = 4.521$, $y = 2.196$.

Algorithm 1 Code structure of VOLNA

```
Initialise mesh from Gmsh file
Initialise state variables
while t < t_final do
    Perform pre-iteration events
    Third-order Runge–Kutta time stepper
        Determine local gradients of state variables on each cell
        Compute a local limiter on each cell
        Reconstruct state variables, compute boundary conditions and determine fluxes across cell faces
        Compute time step
        Apply fluxes and bathymetric source terms to state variables on cells
    Perform post-iteration events
end while
```

with a mathematical formula – were implemented with functionality and simplicity, not performance, in mind.

To better support performance and scalability, and thus allow for large-scale simulations, we have re-engineered the VOLNA code to use OP2 – the overall code structure is kept similar, but matters of data management and parallelism are now entrusted to OP2. To support parallel execution we separated the preprocessing step from the main body of the simulation: first the mesh and simulation parameters are parsed into a HDF5 data file, which can then be read in parallel by the main simulation, which also uses HDF5's parallel file I/O to write results to disk.

Performance-critical parts of the code, essentially any operations on the computational mesh, are re-implemented using OP2: they are written with an element-centric approach and grouped for maximal data reuse. Calculations that were previously a sequence of operations, each calculating all partial results for the entire mesh, now apply only to single elements (such as cells or edges), and OP2 automatically applies these computations to each element – this avoids the use of several temporaries and improves computational density. This process involves outlining the computational kernel to be applied at each set element (cell or edge) to a separate function and writing a call to the OP2 library – a matching code snippet is shown in Fig. 5.

The workflow of VOLNA is made of a few sources of information being created and given as inputs to the code. The first is the merged bathymetry and topography over the whole computational domain, i.e. the sea floor and land elevations,

```
outConservative.H  *= dt;
outConservative.U  *= dt;
outConservative.V  *= dt;

outConservative.H  += MidPointConservative.H;
outConservative.U  += MidPointConservative.U;
outConservative.V  += MidPointConservative.V;

outConservative.H  += inConservative.H;
outConservative.U  += inConservative.U;
outConservative.V  += inConservative.V;

outConservative.H  *= .5;
outConservative.U  *= .5;
outConservative.V  *= .5;

outConservative.H  =
     ( outConservative.H.cwise()  <= EPS )
     .select( EPS, outConservative.H  );
outConservative.Zb  = inConservative.Zb;
ToPhysicalVariables( outConservative, out );
//Implementation of ToPhysicalVariables:
ScalarValue TruncatedH =
     ( outConservative.H.cwise()  < EPS )
     .select( EPS, outConservative.H  );
out.H = outConservative.H;
out.U = outConservative.U.cwise()  / TruncatedH;
out.V = outConservative.V.cwise()  / TruncatedH;
out.Zb  = outConservative.Zb;
```

```
inline void EvolveValuesRK2_2(const  float *dT,
            float *outConservative,
            const float  *inConservative,
            const float  *midPointConservative,
            float *out)
{
  outConservative[0]  *= (*dT);
  outConservative[1]  *= (*dT);
  outConservative[2]  *= (*dT);

  outConservative[0]  += midPointConservative[0];
  outConservative[1]  += midPointConservative[1];
  outConservative[2]  += midPointConservative[2];

  outConservative[0]  += inConservative[0];
  outConservative[1]  += inConservative[1];
  outConservative[2]  += inConservative[2];

  outConservative[0]  *= 0.5f;
  outConservative[1]  *= 0.5f;
  outConservative[2]  *= 0.5f;

  outConservative[0]  = MAX(outConservative[0],EPS);
  outConservative[3]  = inConservative[3];

  //call to ToPhysicalVariables inlined
  float TruncatedH = outConservative[0];
  out[0] = outConservative[0];
  out[1] = outConservative[1] / TruncatedH;
  out[2] = outConservative[2] / TruncatedH;
  out[3] = outConservative[3];
}
...
op_par_loop(EvolveValuesRK2_2, "EvolveValuesRK2_2", cells,
  op_arg_gbl(&dT,1,"float", OP_READ),
  op_arg_dat(outConservative,-1,OP_ID,4,"float",OP_RW),
  op_arg_dat(inConservative,-1,OP_ID,4,"float",OP_READ),
  op_arg_dat(midPointConservative,-1, OP_ID,4,"float",OP_READ),
  op_arg_dat(values_new,-1,OP_ID,4,"float", OP_WRITE));
```

Figure 5. Code snippets from the original and OP2 versions.

over which the flow will propagate. This is given through an unstructured triangular mesh. This is then transformed into a usable input to VOLNA via the *volna2hdf5* code to generate compact HDF5 files. The mesh is also renumbered with the Gibbs–Poole–Stockmeyer algorithm to improve locality.

The second is the dynamic source of the tsunami. It can be an earthquake or a landslide. To describe the temporal evolution of seabed deformation, either a function or a series of files can be used. When a series of files is used (typically when another numerical model provides the spatio-temporal information of a complex deformation), there is a need to define the frequency of these updates in the so-called *vln* generic input file to VOLNA. A recent improvement has been the ability to define these series of files for a sub-region of the computational domain, and at possibly lower resolution. Performance is better when using a function for the seabed deformation since I/O requirements for files can generate large overheads – VOLNA-OP2 allows for describing the initial bathymetry with an input file and then specifying relative deformations using arbitrary code that is a function of spatial coordinates and time. Similarly, one can also define initial conditions for wave elevation and velocity.

The generic input file of VOLNA includes information about the frequency of the updates in the seabed deformation, the virtual gauges in which time series of outputs will be produced, and possibly some options to output time series of outputs over the whole computational domain in order to create movies for instance. These I/O requirements obviously affect performance: the more data to output and the slower the file system, the larger the effect.

To simulate tsunami hazard for a large number of scenarios is computationally expensive, so VOLNA has been replaced in past studies by a statistical emulator, i.e. a cheap surrogate model of the simulator. To build the emulator, input parameters are varied in a design of experiments, and the runs are submitted with these inputs to collect input–output relationships. The output of interest could for example be the waveforms, free surface elevation, and velocity, among others. The increase in flexibility in the definition of the region over which the earthquake source of the tsunami is defined reduces the size of the series of files used as inputs: this is really helpful when a set of simulations needs to be run. Similarly, the ability to specify the relative deformation using an arbitrary code that is a function of spatial coordinates and time also reduces the computational and memory overheads when running a set of simulations.

5 Results

5.1 Running VOLNA

A key goal of this paper is to demonstrate that by utilising the OP2 library VOLNA delivers scalable high performance on a number of common systems. Therefore we take a test case simulating tsunami propagation in the Indian Ocean and run it on three different machines: NVIDIA P100 graphical pro-

cessing units (the Wilkes2 machine in Cambridge's CSD3), a classical CPU architecture in the Peta-5–Skylake part of CSD3 (specifically dual-socket Intel Xeon Gold 6142 16-core Skylake CPUs), and Intel's Xeon Phi platform in Peta-5–KNL (64-core Knights Landing-generation chips, configured in cache mode).

There are five key computational stages that make up 90 % of the total runtime: a stage evolving time using the third-order Runge–Kutta scheme (RK), a "gradients" stage computing gradients among cells, a stage that computes the fluxes across the edges of the mesh ("fluxes"), a stage that computes the minimum time step ("dT"), and a stage that applies the fluxes to the cell-centred state variables ("applyFluxes"). Each of these stages consist of multiple steps, but for performance analysis we study them in groups.

The RK stage is computationally fairly simple (no indirect accesses are made and cell-centred state variables are updated using other cell-centred state variables) and therefore parallelism is easy to exploit, and the limiting factor to performance will be the speed at which we can move data: achieved bandwidth. Both the gradients and the fluxes stages are computationally complex and involve accessing large numbers of data indirectly through cell-to-cell and edge-to-cell mappings. The dT stage moves significant numbers of data to compute the appropriate time step for each cell, triggering an MPI halo exchange as well, and then carries out a global reduction to calculate the minimum – particularly over MPI this can be an expensive operation, but overall it is limited by bandwidth. The applyFluxes stage, while computationally simple, is complex due to its indirect increment access patterns; per-edge values have to be added onto cell-centred values, and in parallelising this operation, OP2 needs to make sure to avoid race conditions. The performance of this loop is limited by the irregular access and control throughout the hardware. For an in-depth study of individual computational loops and their performance, we refer the reader to our previous work in Reguly et al. (2007).

5.2 Tsunami demonstration case

For performance and scaling analysis, we employ the Makran subduction zone as the tsunamigenic source for the numerical simulations. Our region of interest extends from 55 to 79° E and from 6 to 30° N. The bathymetry (Fig. 6a) is obtained from GEBCO (https://www.gebco.net/, last access: 11 November 2018). The region of interest is projected about the centre latitude (i.e. 18° N) to form the rectangular computational domain for VOLNA in Cartesian coordinates (Fig. 6b). This translates to a region of approximately 2500 km × 2700 km in area. The calculation of the sea-floor deformation or uplift (assumed instantaneous) is modelled via the Okada solution (Okada, 1992). This deformation is generated by the earthquake source, which is modelled as a four-segment finite fault model (Table 2) with a uniform slip of 30 m. The non-uniform meshes for the simulation are generated using Gmsh (Geuzaine and Remacle, 2009). A simple strategy is used to generate these meshes. Using the dimensions of the finite fault earthquake sources ($l \times w$), an approximate source wavelength ($\lambda_0 < \min(l, w)$) of the tsunami, and the ocean depth of the Makran trench ($d_0 \sim 3$ km), we calculate the time period (T) of the wave as $T = \frac{\lambda_0}{\sqrt{g d_0}}$. Next, assuming that the time period of the tsunami is the same everywhere in the domain, we get for a depth d_n, $\frac{\lambda_n}{\sqrt{d_n}} = \frac{\lambda_0}{\sqrt{d_0}}$, which in turn relates the characteristic triangle (or element) length h_n for depth d_n as $h_n = \frac{\lambda_0}{k}\sqrt{\frac{d_n}{d_0}}$, where $k = 10$. At the shore (i.e. $d = 0$), a minimum mesh size ($h_{\min}$) is specified. Linear interpolation is carried out to further smoothen the mesh gradation. A combination of λ_0 and $h_{\min}$ is used to generate a series of non-uniform meshes (Table 1 and Fig. 7). We also fix the triangle size as 25 km for regions that are deep inland. Finally, Fig. 8 shows the tsunami waveforms at two virtual gauge locations, from a run on a P100 GPU on CSD3 – the same run on the Peta-5–Skylake cluster gave results with 1.5 % (8–12 June 2015). Simulated time is 21 660 s for all mesh sizes; however, for timed runs at different scales on different platforms we restrict this to 2000 s to conserve computer time.

5.3 Performance and scaling on classical CPUs

As the most commonly used architecture, we first evaluate performance on classical CPUs in the Cambridge CSD3 supercomputer: dual-socket Xeon Gold 6142 CPU, with 16 cores each, supporting the AVX512 instruction set. We test a plain MPI configuration (32 processes per node), as well as a hybrid MPI+OpenMP configuration, with two MPI processes per node (one per socket), and 16 OpenMP threads each, with process and thread binding enabled.

We use OP2's vectorised code generation capabilities, as described in Mudalige et al. (2016). The RK stage performs the same in both variants; however the fluxes and dT stages saw significant performance gains – the compiler did not automatically vectorise computations; it had to be forced to do so. The applyFluxes stage could not be vectorised due to a compiler issue.

On a single node with pure MPI, running the largest mesh, 9 % of time was spent in the RK stage, achieving 182 GB s^{-1} throughput on average; 40 % of time was spent in the gradients stage, achieving 108 GB s^{-1}; 25 % of time was spent in the fluxes stage, achieving 142 GB s^{-1}; 12 % of time was spent in the dT phase, achieving 65 GB s^{-1}; and 12 % of time was spent in the applyFluxes stage, achieving 221 GB s^{-1} thanks to a high degree of data reuse. The maximum bandwidth on this platform is 189 GB s^{-1} as measured by STREAM Triad. The time spent in MPI communications ranged from 23 % on the smallest mesh to 10 % on the largest mesh.

When scaling to multiple nodes with pure MPI, as shown in Fig. 9a, it is particularly evident on the smallest prob-

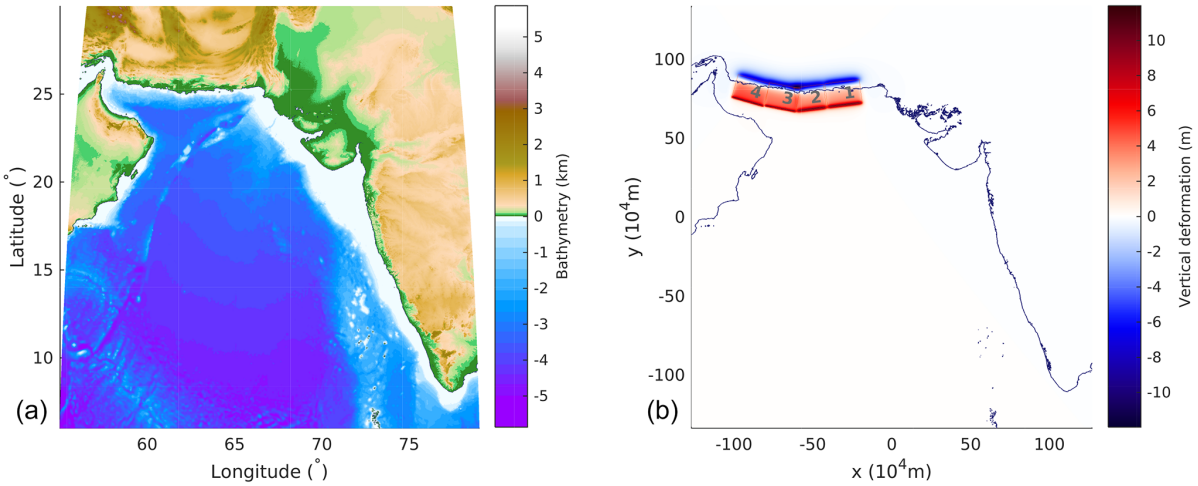

Figure 6. **(a)** Bathymetry from GEBCO's geodetic grid is mapped onto a Cartesian grid for use in VOLNA. **(b)** Uplift caused by a uniform slip of 30 m in the four-segment finite fault model (given in Table 2).

Table 1. Details of the non-uniform (NU) triangular meshes.

Mesh	Name	Vertices n_V	Edges n_E	Triangles n_T	Source λ λ_0	Mesh size at coast h_{min}
NU_0	53.7M	26 863 692	80 564 925	53 701 234	12.5 km	125 m
NU_1	13.8M	6 931 758	20 771 822	13 840 065	25 km	250 m
NU_2	3.6M	1 812 073	5 414 155	3 602 083	50 km	500 m
NU_3	0.95M	485 453	1 435 017	949 565	100 km	1000 m

Table 2. Finite fault parameters of the four-segment tsunamigenic earthquake source.

Segment i	Length (l) (km)	Down-dip width (w) (km)	Longitude (°)	Latitude (°)	Depth (km)	Strike (°)	Dip (°)	Rake (°)
1	220	150	65.23	24.50	10	263	6	90
2	188	150	63.08	24.23	10	263	7	90
3	199	150	61.25	24.00	5	281	8	90
4	209	150	59.32	24.32	5	286	9	90

lem that the problem size per node needs to remain reasonable, otherwise MPI communications will dominate the runtime: for the NU_0 mesh, at 32 nodes 251 s out of 308 s total (81 %). This can be characterised by the strong scaling efficiency: when doubling the number of computational resources (nodes), what percentage of the ideal 2× speedup is achieved. For small node counts these values remain above a reasonable 85 %, but particularly for the smaller problems runtimes actually become worse. It is evident that on the Peta-5–Skylake cluster the interconnect used for MPI communications becomes a bottleneck for scaling – this overhead is significantly lower on Archer, for example; on the largest mesh at 32 nodes this overhead is only 32 %.

We have also evaluated execution with a hybrid MPI+OpenMP approach, as shown with the dashed lines in Fig. 9a. However, on this platform it failed to outperform the pure MPI configuration.

5.4 Performance and scaling on the Intel Xeon Phi

Second, we evaluate Intel's latest many-core chip, the Xeon Phi x7210, which integrates 64 cores, each equipped with AVX-512 vector processing units and supporting four threads and built with a 16 GB on-chip high-bandwidth memory, here used as a cache for off-chip DDR4 memory. The chips were configured in the "quad" mode, with all 16 GB as cache. We evaluate a pure MPI approach (128 processes) as well as using four MPI processes, one per quadrant, and 32 OpenMP threads each. Bandwidth achieved as measured by STREAM Triad is 448 GB s^{-1}. Vectorisation on this platform is paramount for achieving high performance – every

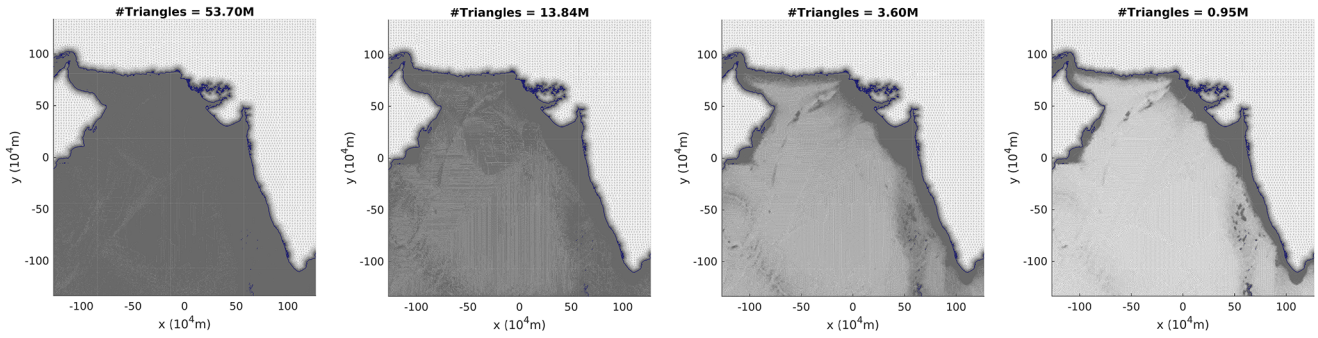

Figure 7. Non-uniform meshes corresponding to the test cases (see Table 1).

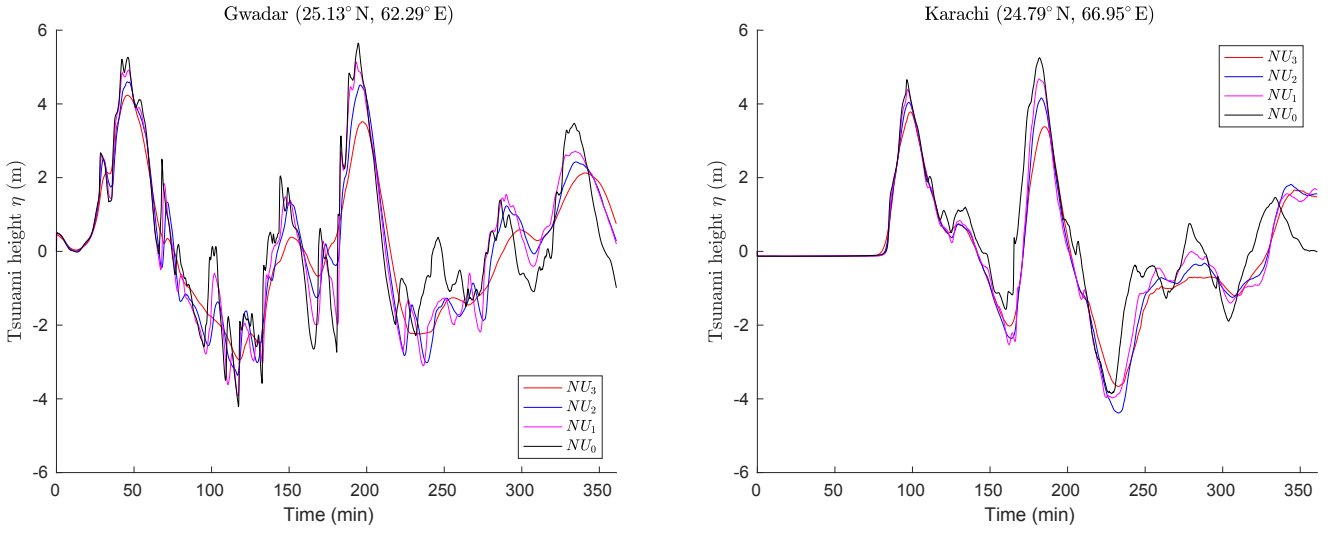

Figure 8. Tsunami waveforms at virtual gauges located at Gwadar and Karachi.

stage with the exception of applyFluxes was vectorised – the latter was not due to compiler issues.

On a single node with pure MPI, the straightforward computations of the RK stage can utilise the available high bandwidth very efficiently: only 8.3 % of time spent here, achieving 194 GB s^{-1}. The gradients stage takes 42 % of the time, achieving 82 GB s^{-1}; the fluxes stage takes 25 % of the time and achieves 104 GB s^{-1}; dT takes 11.4 % and achieves 46 GB s^{-1}; and the applyFluxes stage takes 11.6 % and achieves 165 GB s^{-1}. On the largest mesh, the Zeon Phi system is 21 % slower than a single node of the classical CPU system.

Performance when scaling to multiple nodes with pure MPI is shown in Fig. 9b: it is quite clear that scaling is worse than on the classical CPU architecture for smaller problem sizes – the Xeon Phi requires a considerably larger problem size per node to operate efficiently. Strong scaling efficiency is particularly poor on the smallest mesh, but even on the largest mesh it is only between 63 % and 92 %. Similar to the classical CPU system, the interconnect becomes a bottleneck to scaling. Running with a hybrid MPI+OpenMP configuration on the Xeon Phi does improve scaling significantly, as shown in Fig. 9b – this is due to having to exchange much fewer (but larger) messages. Strong scaling efficiency on the largest problem remains above 82 %. At scale, at least on this cluster, the Xeon Phi can outperform the classical CPU system on a node-to-node basis of comparison.

5.5 Performance and scaling on P100 GPUs

Third, we evaluate performance on GPUs – an architecture that has continually been increasing its market share in high-performance computing thanks to its efficient parallel architecture. The P100 GPUs are hosted in the Wilkes2 system, with 4 GPUs per node connected via the PCI-e bus. Each chip contains 60 scalar multiprocessors, with 64 CUDA cores each, giving a total of 3840 cores. There is also 16 GB of high-bandwidth memory on package, with a bandwidth of 497 GB s^{-1}. To utilise these devices, we use CUDA code generated by OP2 and compiled with CUDA 9. Similar to

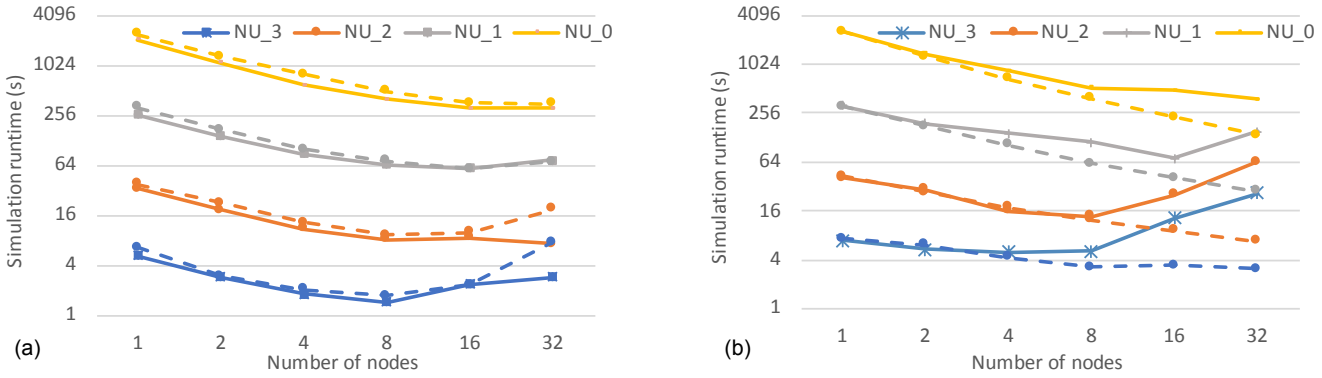

Figure 9. Performance scaling on **(a)** Peta-5–CPU (Intel Xeon CPU) and **(b)** Peta-5–KNL (Intel Xeon Phi) at different mesh sizes with pure MPI (solid) and MPI+OpenMP (dashed).

Intel's Xeon Phi, high vector efficiency is required for good performance on the GPU.

On a single GPU, running the second-largest mesh NU_1 (because NU_0 does not fit in memory), 8.3 % of runtime is spent in the RK stage, achieving 342 GB s^{-1}; gradients takes 50 % of the time, achieving only 136 GB s^{-1} due to its high complexity; fluxes takes 15 %, achieving 379 GB s^{-1} thanks to a high degree of data reuse in indirect accesses; dT takes 4.4 % and achieves 382 GB s^{-1}; and finally applyFluxes takes 20 % of the time, achieving 204 GB s^{-1}. Indeed, this last phase has the most irregular memory access patterns, which is commonly known to degrade performance on GPUs. Nevertheless, even a single GPU outperforms a classical CPU node by a factor of 1.5, and the Xeon Phi by 1.85.

Performance when scaling to multiple GPUs is shown in Fig. 10; similar to the Xeon Phi, GPUs are also sensitive to the problem size and the overhead of MPI communications. However, given that there are four GPUs in one node, the overhead of communications is significantly lower. For the smallest problem, efficiency drops from 78 % to 58 %, and for the largest problem efficiency drops from 95 % to 89 %. Thanks to much better scaling (due to lower MPI overhead), 32 GPUs are 6.9/2.6× faster than 32 nodes of Xeon CPUs and Xeon Phi.

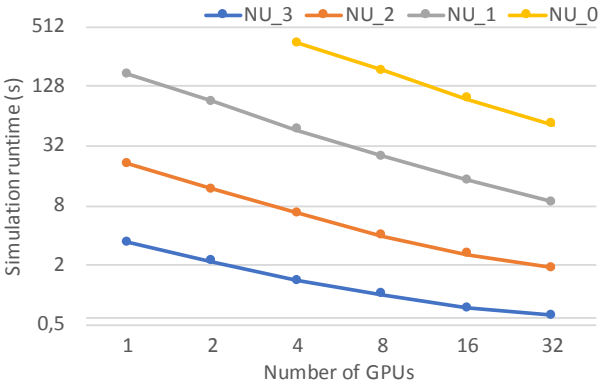

Figure 10. Performance scaling on Wilkes2 (P100 GPU) at different mesh sizes.

5.6 Running costs and power consumption

Ultimately, when one needs to decide what platform to run these simulations on, a key deciding factor aside from time to solution is cost to solution. In the analysis above, aside from discussing absolute performance metrics, we have reported speedup numbers relative to other platforms – which from a performance benchmarking perspective is not strictly fair. However, such relative performance figures combined with the cost of access do help in the decision.

Admittedly the cost of buying hardware as well as the cost of core hours or GPU hours varies significantly; therefore here we do not look at specific prices. However, energy consumption is an indicator of pricing. A dual-socket CPU consumes up to 260 W, which is then roughly tripled when looking at the whole node due to memory, disks, networking, etc. In comparison, the Intel Xeon Phi CPU has a thermal design power of 215 W, roughly 750 W for the node. A P100 GPU has a TDP of 300 W, but has to be hosted in a CPU system – the more GPUs in a single machine, the better amortised this cost is: the TDP of a GPU node in Wilkes2 is around 1.8 kW (4×250 for the GPUs, plus 800 (Ithaca, New York, USA) for the rest of the system) – which averages to 450 W GPU^{-1}. Thus in terms of power efficiency GPUs are by far the best choice for VOLNA. Nevertheless, a key benefit of VOLNA-OP2 is that it can efficiently utilise any high-performance hardware commonly available.

6 Conclusions

In this paper we have introduced and described the VOLNA-OP2 code; a tsunami simulator built on the OP2 library, enabling execution on CPUs, GPUs, and heterogeneous super-

computers. By building on OP2, the science code of VOLNA itself is written only once using a high-level abstraction, capturing what to compute but not how to compute it. This approach enables OP2 to take control of the data structures and parallel execution; VOLNA is then automatically translated to use sequential execution, OpenMP, or CUDA, and by linking with the appropriate OP2 back-end library, these are then combined with MPI. This approach also future-proofs the science code: as new architectures come along, the developers of OP2 will update the back-ends and the code generators, allowing VOLNA to make use of them without further effort. This kind of ease of use and portability makes VOLNA-OP2 unique among tsunami simulation codes. Through performance scaling and analysis of the code on traditional CPU clusters, as well as GPUs and Intel's Xeon Phi, we have demonstrated that VOLNA-OP2 indeed delivers high performance on a variety of platforms and, depending on problem size, scales well to multiple nodes.

We have described the key features of VOLNA, the discretisation of the underlying physical model (i.e. NSWE) in the finite-volume context and the third-order Runge–Kutta time stepper, as well as the input–output features that allow the integration of the simulation step into a larger workflow; initial conditions, and bathymetry in particular, can be specified in a number of ways to minimise I/O requirements, and parallel output is used to write out simulation data on the full mesh or specified points.

There is still a need for even more streamlined and efficient workflows. For instance, we could integrate the finite fault source model for the slip with some assumptions on the rupture dynamics within VOLNA. We could also integrate the bathymetry-based meshing (the mesh needs to be tailored to the depth and gradients of the bathymetry to optimally reduce computational time). Indeed, there would be even fewer exchanges of files and more efficient computations, especially in the context of uncertainty quantification tasks such as emulation or inversion.

In the end, the gain in computational efficiency will allow higher-resolution modelling, such as using 2 m topography and bathymetry collected from lidar, i.e. a greater capability. It will allow greater capacity by enabling more simulations to be performed. Both of these enhancements will subsequently lead to better warnings more tailored to the actual impact on the coast as well as better urban planning since hazard maps will gain in precision geographically and probabilistically due to the possibility of exploring a larger number of more realistic scenarios.

Code availability. The code is available at https://github.com/reguly/volna/ (last access: 11 November 2018) and https://doi.org/10.5281/zenodo.1413124 (Reguly et al., 2018). It depends on the OP2 library, which is also available at https://github.com/OP-DSL/OP2-Common (last access: 11 November 2018), and depends on an MPI distribution, parallel HDF5, and a partitioner, such as ParMETIS or PT-Scotch. For GPU execution, the CUDA SDK and a compatible device are required.

Author contributions. All authors contributed to the writing of the paper. IZR performed the majority of coding, with all other co-authors contributing with extensions and the numerical aspects of the code. DG and DG designed and evaluated the test cases. IZR ran the scalability studies on various supercomputers. MBG, SG, and FD provided overall supervision of the code design, evaluation, and writing.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. We would like to thank Endre László, formerly of PPCU ITK, who worked on the initial port of VOLNA to OP2. István Z. Reguly was supported by the János Bólyai Research Scholarship of the Hungarian Academy of Sciences. Project no. PD 124905 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the PD_17 funding scheme. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work (https://doi.org/10.5281/zenodo.22558). Serge Guillas gratefully acknowledges support through the NERC grants PURE (Probability, Uncertainty and Risk in the Natural Environment) NE/J017434/1 and "A demonstration tsunami catastrophe risk model for the insurance industry" NE/L002752/1. Serge Guillas and Devaraj Gopinathan acknowledge support from the NERC project (NE/P016367/1) under the Global Challenges Research Fund: Building Resilience programme. Devaraj Gopinathan acknowledges support from the Royal Society, UK, and Science and Engineering Research Board (SERB), India, for the Royal Society–SERB Newton International Fellowship (NF151483). Daniel Giles acknowledges support by the Irish Research Council's Postgraduate Scholarship Programme.

Edited by: Simone Marras
Reviewed by: two anonymous referees

References

Abadie, S. M., Harris, J. C., Grilli, S. T., and Fabre, R.: Numerical modeling of tsunami waves generated by the flank collapse of the Cumbre Vieja Volcano (La Palma, Canary Islands): Tsunami source and near field effects, J. Geophys. Res.-Oceans, 117, C05030, https://doi.org/10.1029/2011JC007646, 2012.

Acuña, M. and Aoki, T.: Real-time tsunami simulation on multi-node GPU cluster, in: ACM/IEEE conference on supercomputing, 14–20 November 2009, Portland, Oregon USA, 2009.

Anita, G., Andrey, B., Ana, B. M., Jörn, B., Antonio, C., Gareth, D., L., G. E., Sylfest, G., I., G. F., Jonathan, G., B., H. C., J., L. R., Stefano, L., Finn, L., Rachid, O., Christof, M., Raphaël, P., Tom, P., Jascha, P., William, P., Jacopo, S., B., S. M., and Kie, T. H.: Probabilistic Tsunami Hazard Analysis: Multiple

Sources and Global Applications, Rev. Geophys., 55, 1158–1198, https://doi.org/10.1002/2017RG000579, 2017.

Barth, T. and Jespersen, D.: The design and application of upwind schemes on unstructured meshes, American Institute of Aeronautics and Astronautics, https://doi.org/10.2514/6.1989-366, 1989.

Beck, J. and Guillas, S.: Sequential Design with Mutual Information for Computer Experiments (MICE): Emulation of a Tsunami Model, SIAM/ASA Journal on Uncertainty Quantification, 4, 739–766, https://doi.org/10.1137/140989613, 2016.

Behrens, J. and Dias, F.: New computational methods in tsunami science, Philos. T. R. Soc. A, 373, 20140382, https://doi.org/10.1098/rsta.2014.0382, 2015.

Bernard, E., Mofjeld, H., Titov, V., Synolakis, C., and González, F.: Tsunami: scientific frontiers, mitigation, forecasting and policy implications, Philos. T. R. Soc. A, 364, 1989–2007, https://doi.org/10.1098/rsta.2006.1809, 2006.

Brodtkorb, A. R., Hagen, T. R., Lie, K.-A., and Natvig, J. R.: Simulation and visualization of the Saint-Venant system using GPUs, Computing and Visualization in Science, 13, 341–353, https://doi.org/10.1007/s00791-010-0149-x, 2010.

Castro, M., González-Vida, J., Macías, J., Ortega, S., and de la Asunción, M.: Tsunami-HySEA: a GPU-based model for tsunami early warning systems, in: Proc XXIV Congress on Differential Equations and Applications, June, Cádiz, Spain, 8–12 June 2015.

Davies, G., Griffin, J., Løvholt, F., Glimsdal, S., Harbitz, C., Thio, H. K., Lorito, S., Basili, R., Selva, J., Geist, E., and Baptista, M. A.: A global probabilistic tsunami hazard assessment from earthquake sources, Geol. Soc. Spec. Publ., 456, 219–244, https://doi.org/10.1144/SP456.5, 2017.

Dias, F., Dutykh, D., O'Brien, L., Renzi, E., and Stefanakis, T.: On the Modelling of Tsunami Generation and Tsunami Inundation, Procedia IUTAM, 10, Mechanics for the World: Proceedings of the 23rd International Congress of Theoretical and Applied Mechanics, ICTAM2012, 19–24 August 2012, Beijing, China, 338–355, https://doi.org/10.1016/j.piutam.2014.01.029, 2014.

Dutykh, D. and Dias, F.: Tsunami generation by dynamic displacement of sea bed due to dip-slip faulting, Math. Comput. Simulat., 80, 837–848, https://doi.org/10.1016/j.matcom.2009.08.036, 2009.

Dutykh, D., Poncet, R., and Dias, F.: The VOLNA code for the numerical modeling of tsunami waves: Generation, propagation and inundation, Eur. J. Mech. B-Fluid., 30, 598–615, 2011.

Gailler, A., Hébert, H., Loevenbruck, A., and Hernandez, B.: Simulation systems for tsunami wave propagation forecasting within the French tsunami warning center, Nat. Hazards Earth Syst. Sci., 13, 2465–2482, https://doi.org/10.5194/nhess-13-2465-2013, 2013.

Geist, E. L. and Parsons, T.: Probabilistic Analysis of Tsunami Hazards, Nat. Hazards, 37, 277–314, https://doi.org/10.1007/s11069-005-4646-z, 2006.

George, D. L. and LeVeque, R. J.: Finite volume methods and adaptive refinement for global tsunami propagation and local inundation, Science of Tsunami Hazards, 24, 319–328, 2006.

Geuzaine, C. and Remacle, J.-F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Meth. Eng., 79, 1309–1331, https://doi.org/10.1002/nme.2579, 2009.

Giles, M. B., Mudalige, G. R., Sharif, Z., Markall, G., and Kelly, P. H.: Performance analysis and optimization of the OP2 framework on many-core architectures, Comput. J., 55, 168–180, 2011.

Gisler, G., Weaver, R., and Gittings, M. L.: SAGE calculations of the tsunami threat from La Palma, Sci. Tsunami Hazards, 24, 288–312, 2006.

Glimsdal, S., Pedersen, G. K., Harbitz, C. B., and Løvholt, F.: Dispersion of tsunamis: does it really matter?, Nat. Hazards Earth Syst. Sci., 13, 1507–1526, https://doi.org/10.5194/nhess-13-1507-2013, 2013.

Goda, K., Mai, P. M., Yasuda, T., and Mori, N.: Sensitivity of tsunami wave profiles and inundation simulations to earthquake slip and fault geometry for the 2011 Tohoku earthquake, Earth Planets Space, 66, 105, https://doi.org/10.1186/1880-5981-66-105, 2014.

Gopinathan, D., Venugopal, M., Roy, D., Rajendran, K., Guillas, S., and Dias, F.: Uncertainties in the 2004 Sumatra-Andaman source through nonlinear stochastic inversion of tsunami waves, P. Roy. Soc. Lond. A Mat., 473, 20170353, https://doi.org/10.1098/rspa.2017.0353, 2017.

Harig, S., Pranowo, W. S., and Behrens, J.: Tsunami simulations on several scales, Ocean Dynam., 58, 429–440, 2008.

Jacobs, C. T. and Piggott, M. D.: Firedrake-Fluids v0.1: numerical modelling of shallow water flows using an automated solution framework, Geosci. Model Dev., 8, 533–547, https://doi.org/10.5194/gmd-8-533-2015, 2015.

Kennedy, A. B., Chen, Q., Kirby, J. T., and Dalrymple, R. A.: Boussinesq modeling of wave transformation, breaking, and runup. I: 1D, J. Waterw. Port, C., 126, 39–47, 2000.

Kervella, Y., Dutykh, D., and Dias, F.: Comparison between three-dimensional linear and nonlinear tsunami generation models, Theor. Comp. Fluid Dyn., 21, 245–269, https://doi.org/10.1007/s00162-007-0047-0, 2007.

Lawrence, B. N., Rezny, M., Budich, R., Bauer, P., Behrens, J., Carter, M., Deconinck, W., Ford, R., Maynard, C., Mullerworth, S., Osuna, C., Porter, A., Serradell, K., Valcke, S., Wedi, N., and Wilson, S.: Crossing the chasm: how to develop weather and climate models for next generation computers?, Geosci. Model Dev., 11, 1799–1821, https://doi.org/10.5194/gmd-11-1799-2018, 2018.

Lay, T., Kanamori, H., Ammon, C. J., Nettles, M., Ward, S. N., Aster, R. C., Beck, S. L., Bilek, S. L., Brudzinski, M. R., Butler, R., DeShon, H. R., Ekström, G., Satake, K., and Sipkin, S.: The Great Sumatra-Andaman Earthquake of 26 December 2004, Science, 308, 1127–1133, https://doi.org/10.1126/science.1112250, 2005.

Liang, W.-Y., Hsieh, T.-J., Satria, M. T., Chang, Y.-L., Fang, J.-P., Chen, C.-C., and Han, C.-C.: A GPU-Based Simulation of Tsunami Propagation and Inundation, Springer Berlin Heidelberg, 593–603, https://doi.org/10.1007/978-3-642-03095-6_56, 2009a.

Liang, W.-Y., Hsieh, T.-J., Satria, M. T., Chang, Y.-L., Fang, J.-P., Chen, C.-C., and Han, C.-C.: A GPU-Based Simulation of Tsunami Propagation and Inundation, in: Algorithms and Architectures for Parallel Processing, edited by: Hua, A. and Chang, S.-L., Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 593–603, 2009b.

Liu, P. L., Woo, S.-B., and Cho, Y.-S.: Computer programs for tsunami propagation and inundation, Cornell University, Ithaca, New York, USA, 1998.

Liu, X. and Guillas, S.: Dimension Reduction for Gaussian Process Emulation: An Application to the Influence of Bathymetry on Tsunami Heights, SIAM/ASA Journal on Uncertainty Quantification, 5, 787–812, https://doi.org/10.1137/16M1090648, 2017.

Løvholt, F., Pedersen, G., Harbitz, C. B., Glimsdal, S., and Kim, J.: On the characteristics of landslide tsunamis, Philos. T. R. Soc. A, 373, 20140376, https://doi.org/10.1098/rsta.2014.0376, 2015.

Lynett, P. J., Wu, T.-R., and Liu, P. L.-F.: Modeling wave runup with depth-integrated equations, Coast. Eng., 46, 89–107, 2002.

Macías, J., Castro, M. J., Ortega, S., Escalante, C., and González-Vida, J. M.: Performance Benchmarking of Tsunami-HySEA Model for NTHMP's Inundation Mapping Activities, Pure Appl. Geophys., 174, 3147–3183, https://doi.org/10.1007/s00024-017-1583-1, 2017.

Mudalige, G., Giles, M., Reguly, I., Bertolli, C., and Kelly, P.: OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures, in: Innovative Parallel Computing (InPar), San Jose, California, USA, 1–12, IEEE, 2012.

Mudalige, G. R., Reguly, I. Z., and Giles, M. B.: Auto-vectorizing a Large-scale Production Unstructured-mesh CFD Application, in: Proceedings of the 3rd Workshop on Programming Models for SIMD/Vector Processing, WPMVP '16, 12–16 March 2016, Barcelona, Spain, ACM, New York, NY, USA, 5:1–5:8, https://doi.org/10.1145/2870650.2870651, 2016.

O'Brien, L., Christodoulides, P., Renzi, E., Stefanakis, T., and Dias, F.: Will oscillating wave surge converters survive tsunamis?, Theor. Appl., 5, 160–166, 2015.

Okada, Y.: Internal deformation due to shear and tensile faults in a half-space, B. Seismol. Soc. Am., 82, 1018–1040, 1992.

Poncet, R., Campbell, C., Dias, F., Locat, J., and Mosher, D.: A Study of the Tsunami Effects of Two Landslides in the St. Lawrence Estuary, Springer Netherlands, Dordrecht, the Netherlands, 755–764, https://doi.org/10.1007/978-90-481-3071-9_61, 2010.

Reguly, I. Z., László, E., Mudalige, G. R., and Giles, M. B.: Vectorizing Unstructured Mesh Computations for Many-core Architectures, in: Proceedings of Programming Models and Applications on Multicores and Manycores, PMAM'14, 15–19 February 2014, Orlando, FL, USA, ACM, New York, NY, USA, 39:39–39:50, https://doi.org/10.1145/2578948.2560686, 2007.

Reguly, I. Z., Giles, D., Gopinathan, D., Quivy, L., Beck, J. H., Giles, M. B., Guillas, S., and Dias, F.: The Volna-OP2 software, version 1.5, Zenodo, https://doi.org/10.5281/zenodo.1413124, 2018.

Richards, A.: University of Oxford Advanced Research Computing, Zenodo, https://doi.org/10.5281/zenodo.22558, 2015.

Salmanidou, D. M., Guillas, S., Georgiopoulou, A., and Dias, F.: Statistical emulation of landslide-induced tsunamis at the Rockall Bank, NE Atlantic, P. Roy. Soc. Lond. A Mat., 473, 20170026, https://doi.org/10.1098/rspa.2017.0026, 2017.

Salmanidou, D. M., Georgiopoulou, A., Guillas, S., and Dias, F.: Rheological considerations for the modelling of submarine sliding at Rockall Bank, NE Atlantic Ocean, Phys. Fluids, 30, 030705, https://doi.org/10.1063/1.5009552, 2018.

Satria, M. T., Huang, B., Hsieh, T.-J., Chang, Y.-L., and Liang, W.-Y.: GPU acceleration of tsunami propagation model, IEEE J. Sel. Top. Appl., 5, 1014–1023, 2012.

Schling, B.: The Boost C++ Libraries, XML Press, Amsterdam, Netherlands, 2011.

Stefanakis, T. S., Contal, E., Vayatis, N., Dias, F., and Synolakis, C. E.: Can small islands protect nearby coasts from tsunamis? An active experimental design approach, Proc. R. Soc. A, 470, 20140575, https://doi.org/10.1098/rspa.2014.0575, 2014.

Synolakis, C. E.: The runup of solitary waves, J. Fluid Mech., 185, 523–545, https://doi.org/10.1017/S002211208700329X, 1987.

Tavakkol, S. and Lynett, P.: Celeris: A GPU-accelerated open source software with a Boussinesq-type wave solver for real-time interactive simulation and visualization, Comput. Phys. Commun., 217, 117–127, https://doi.org/10.1016/j.cpc.2017.03.002, 2017.

The HDF Group: Hierarchical data format version 5, available at: http://www.hdfgroup.org/HDF5 (last access: 11 November 2018), 2000–2010.

Titov, V. V. and Gonzalez, F. I.: Implementation and testing of the method of splitting tsunami (MOST) model, Tech. rep., NOAA Technical Memorandum ERL PMEL-112, 11 pp. UNIDATA, Washington, D.C., USA, 1997.

Vater, S. and Behrens, J.: Well-balanced inundation modeling for shallow-water flows with discontinuous Galerkin schemes, in: Finite volumes for complex applications VII-elliptic, parabolic and hyperbolic problems, Springer, Berlin, 965–973, 2014.

Yusuke, O., Fumihiko, I., and Daisuke, S.: Near-field tsunami inundation forecast using the parallel TUNAMI-N2 model: Application to the 2011 Tohoku-Oki earthquake combined with source inversions, Geophys. Res. Lett., 42, 1083–1091, https://doi.org/10.1002/2014GL062577, 2014.

Zhang, Y. J. and Baptista, A. M.: An Efficient and Robust Tsunami Model on Unstructured Grids. Part I: Inundation Benchmarks, Pure Appl. Geophys., 165, 2229–2248, https://doi.org/10.1007/s00024-008-0424-7, 2008.